Project A5 Task 3 – Tony Ngo

**Task 3A**

1. **What are the basic steps (show all steps) in building a parallel program? Show at least one example.**

The first step is figuring out which tasks can be running parallel with one another. Then it splits the workload into a master/worker. The master creates the array and splits it amongst the available workers, then sends each worker into its own subarray, and then receives the computation from each worker. The worker performs the processing on the subarray. An example of this would be how to find the value for pi.

1. **What is MapReduce?**

MapReduce is a way to processing big data sets with a parallel and distributed algorithm on a cluster.

1. **What is map and what is reduce?**

Map is a function that creates a key/value pair to generate a set of intermediate key/value pairs, then the reduce function merges all of the intermediate values that are associated with the same key.

1. **Why MapReduce?**

Compared to other similar structures (i.e. MPI), MapReduce has built in fault tolerance, which means that a system is still able to operate

1. **Show an example for MapReduce.**

A picture containing text, map

Description automatically generated

1. **Explain in your own words how MapReduce model is executed?**The allocated memory given implements a master, then the master separates the memory into different sections called the workers who are given functions Map and Reduce. Once given their instructions, the worker creates a key/value pair and an intermediate key/value pair is created by the Map Workers. Then the pairs created are written to the local disk and given locations; the locations are given to the master who has to give these locations to the workers. Once the Reduce Worker is given the location, it reads the data from the intermediate keys and sorts them in a way that all same keys are together. Once all of these tasks are completed, the master calls the user program and returns the code.
2. **List and describe three examples that are expressed as MapReduce computations.**

Count of URL Access Frequency: The map function processes logs of who wants to access the webpage and outputs the value <URL,1>, then the reduce function adds all values of the same URL together and gives the result <target, list(source)>.

Distributed Grep: The map function will output a line if it matches a given pattern, then the reduce function will copy the intermediate data into the output.

Reverse Web-Link Graph: The map function shows <target, source> pairs for each link to a target URL from the source. The reduce function takes the list of source URLs with a target URL and outputs the pair: <target, list(source)>.

1. **When do we use OpenMP, MPI and, MapReduce (Hadoop), and why?**

OpenMP: We use OpenMP when we want to introduce shared memory parallelism in our code. It serves as a library that enables multithreading/parallel programming.

MPI: Message Passing Interface is used to develop parallel applications, specifically for science, because they can implement load balance well.

MapReduce: We use MapReduce to reduce the amount of data used in operations and reduce complex problems.

1. **In your own words, explain what a Drug Design and DNA problem is in no more than 150 words.**

When drugs are designed, they find ligands, new pieces, to change the shape of a protein. Once a new ligand is found, they compute a “score” for it, which is calculated based on: how well it fits that protein and produces a desired shape. Then, all new ligands are compared, and they identify the ones with the highest score. The issue is computing the score takes a long time; a way to reduce this is using parallelism in the computational thread for different ligands.

**Task 3B**

**A screenshot of a social media post

Description automatically generated**

This is FileZilla, the application that I used to transfer the files from my MacBook to my Raspberry PI.

2.2 Compilation:

A screen shot of a computer

Description automatically generated

I did not run into any issues while creating the 2.2 compilation. However, trying to get the files from my MacBook onto my Raspberry PI was an issue. Through various types of troubleshooting, how I did it was I used FileZilla to transfer the downloaded file from iCollege onto my PI. I created the folder using the command “mkdir sequential” and then copied the folder into there.

3.2 Compilation:

A screen shot of a computer

Description automatically generated

I created the folder using the command “mkdir openmp1”. I transferred the files “openmp1.cpp” and “MakeFile” from my MacBook to the PI using FileZilla. For the 3.2 compilation, I was running to a fatal error where it stated, “**fatal error:** ttb.h: No such file or directory”. I tried using the command “g++ -o dd\_omp dd\_omp.cpp” to alleviate it but I still came across the issue. After some troubleshooting, to fix this error, I had to install the ttb library using the command, “sudo apt-get install libtbb-dev”. Once I did that I was able to run the “make” command to create dd\_omp.

4.1 Compilation:

A picture containing drawing

Description automatically generated

I created the folder using the command “mkdir cplusthreads1”. I transferred the files “dd\_threads.cpp” and “Makefile” from my Macbook to the PI using FileZilla. For the compilation, it went seamlessly because I was able to fix the fatal error by downloading the tbb.h library in the 3.1 compilation.

**5**

|  |  |
| --- | --- |
| **Implementation** | **Time(s)** |
| dd\_serial | 125.53 |
| dd\_omp | 0.02 |
| dd\_threads | 0.02 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Implementation** | **Time(s) 2 Threads** | **Time(s) 3 Threads** | **Time(s) 4 Threads** |
| dd\_omp | 0.02 | 0.04 | 0.22 |
| dd\_threads | 0.02 | 0.04 | 0.15 |

**2.3 – Discussion Questions**

1. **Which approach is the fastest?**

The fastest approach for each is using the C++11 thread solutions

1. **Determine the number of lines in each file (use wc -l). How does the C++11 implementation compare to the OpenMP implementations?**

Sequential Lines: 170

OpenMP Lines: 193

C++11 Lines: 207

1. **Increase the number of threads to 5 threads. What is the run time for each?**

OpenMP Real Time (s): 0.95

C++11 Real Time (s): 0.68

1. **Increase the maximum ligand length to 7 and rerun each program. What is the run time for each?**

OpenMP Real Time (s): 73.91

C++11 Real Time (s): 41.66